HKN CS61C Final Review

Fall 2014

Riyaz Faizullabhoy
Eldon Schoop
Vikram Sreekanti

Hello!

Since the presenters are not all affiliated with the course, the standard disclaimer applies: This review session is not officially endorsed by the course staff.

That said, we hope you find our review pointers (haha) helpful!

Also, please fill out a feedback form on your way out

Agenda

- Midterm Potpourri
  - Caches, Associativity
- VM
- Parallelism
  - WSC, Map Reduce, Spark
  - SSE
  - openmp
- Digital Systems
  - Combinational logic, FSMs
  - MIPS CPU

We are **not** going to cover a lot of other concepts you still probably need to know:

C, Pipelining, I/O, Floating point, CALL, MIPS, Number rep.

Cache Money

(Adopted from Sung Roa's Fall 2013 slides) <3

What's the issue with the direct mapped caches?

A0	A1	A2	A3

Let's say you are trying to access memory locations 0b10010 and 0b00001 back and forth. How would this 4x4 direct mapped cache handle it?

First access is a miss!

To load data B, you must remove A, since they have the same index & different tag!

A = 0b10010, B= 0b00001

A = 0b1 00 10, B = 0b0 00 01

Desired access:

A, B, A, B, …

What’s the issue with the direct mapped caches?

B0	B1	B2	B3

Let’s say you are trying to access memory locations 0b10010 and 0b00001 back and forth. How would this 4x4 direct mapped cache handle it?

A = 0b10010, B= 0b00001

A = 0b1 00 10, B = 0b0 00 01

Desired access:

A, B, A, B, …

First access is a miss!

To load data B, you must remove A, since they have the same index & different tag!

Second access is a miss!

When you try to access A again, it’s no longer in the cache!

What's the issue with the direct mapped caches?

0 Percent Hit Rate for Simple Case!

A0	A1	A2	A3

Let's say you are trying to access memory locations 0b10010 and 0b00001 back and forth. How would this 4x4 direct mapped cache handle it?

A = 0b10010, B= 0b00001

A = 0b1 00 10, B = 0b0 00 01

Desired access:

A, B, A, B, …

First access is a miss!

To load data B, you must remove A, since they have the same index & different tag!

Second access is a miss!

When you try to access A again, it's no longer in the cache!

Third access is a miss!

How do we get around that problem?

Instead of having one direct mapped cache, have multiple direct mapped caches!

A0	A1	A2	A3

B0	B1	B2	B3

First access is a miss!

Second access is a miss!

When you try to access A again, it's still in the cache!

Third access is a hit!

When you try to access A again, it's still in the cache!

Fourth access is a hit!

Fifth access is a hit!

So on…

Let's try to make the same memory accesses again!

A = 0b10010, B= 0b00001

A = 0b10 0 10, B = 0b00 0 01

A, B, A, B, …

~100 Percent Hit Rate if continued for extended period of time!

Details of implementation

As we have limited cache space, when we give you a total cache size and the number of sets, the number of sets will directly affect the T-I-O of the cache.

For example, if you have a 4KiB 4-way set-associative cache with block size of 64B, with 16 address bits:

Since the block size is 64B, equivalent to 2^6B, you know that the number of offset bits is 6.

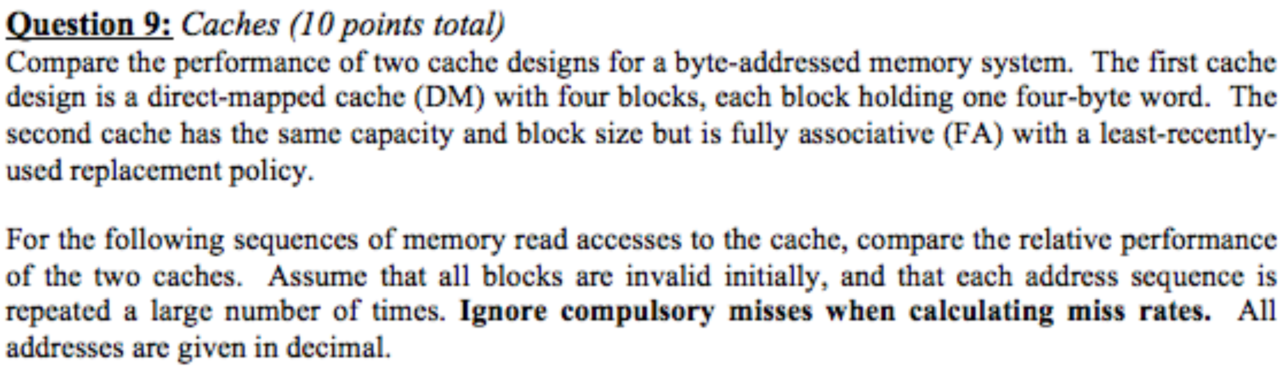
You have 1KiB per direct mapped cache (as it is 4-way set associative), which means that you have 2^10B/2^6B number of rows.

Since you have 2^4 number of rows, you know that the number of index bits is 4.

Since you know that the address is 16 bits, offset is 6 bits, and index is 4 bits, you know that the number of tag bits is 16 – 6 – 4 = 6 bits.

Consider Set-Associative Caches as a collection of direct mapped caches!

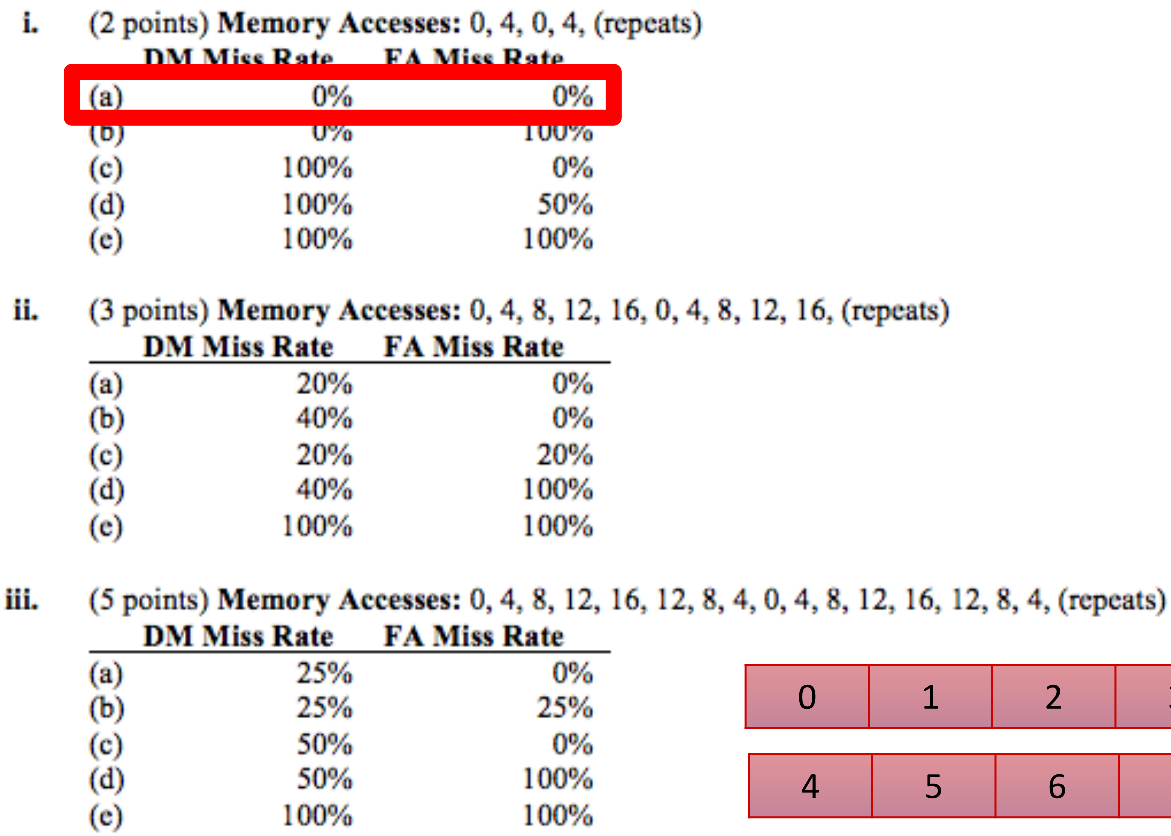
Example Problem (Fa 2012)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i) Memory Accesses:
Direct: 000, 100, 000, 100, …
M, M, H, H, H, H, …
FA: 000, 100, 000, 100, …
M, M, H, H, H, H, …



i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
(d)	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
(b)	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

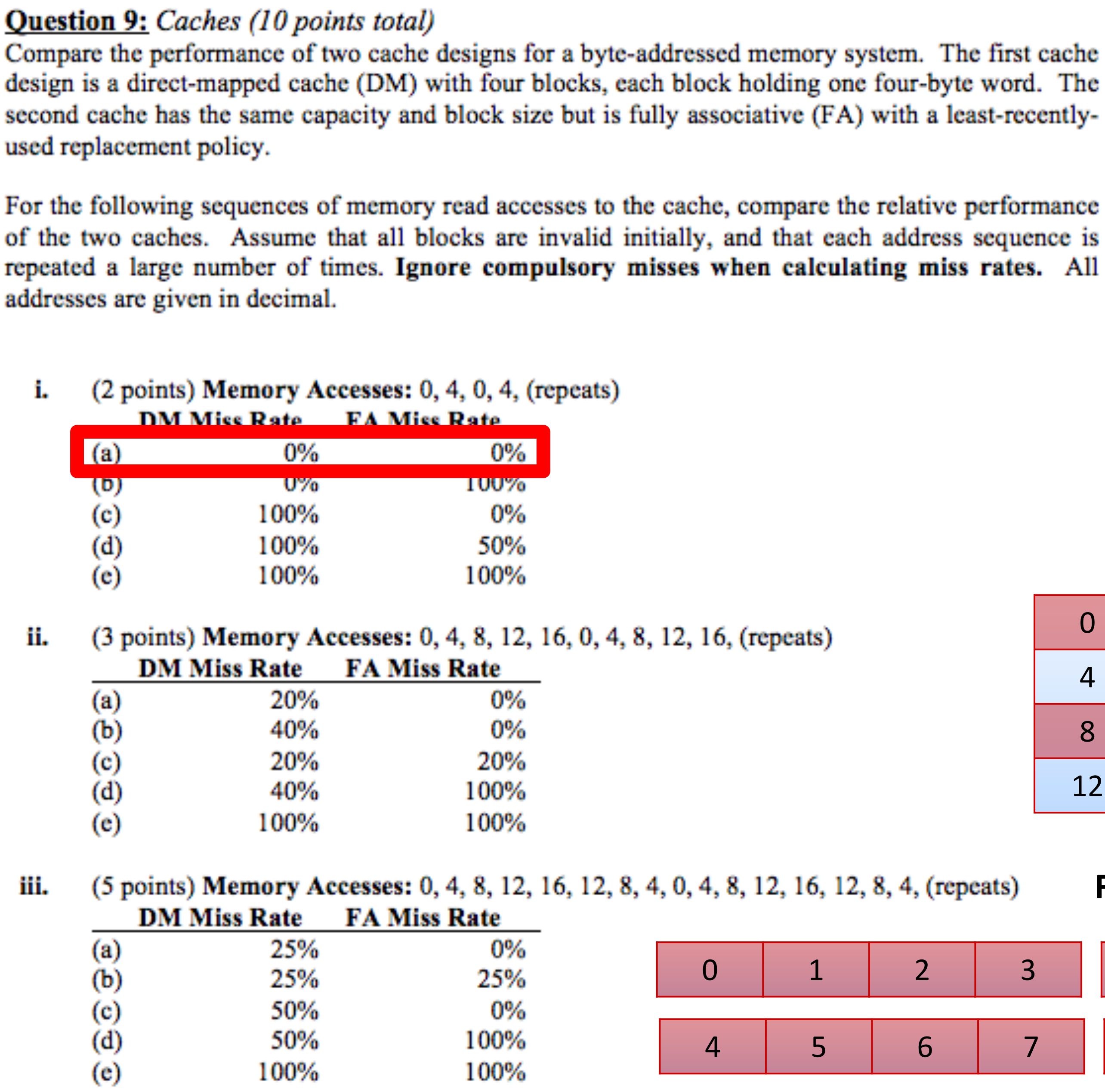
Direct

0	1	2	3
4	5	6	7

FA:

0	1	2	3				
4	5	6	7				

Example Problem (Fa 2012)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
(d)	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
(b)	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

ii) Memory Accesses:
D: 00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…

FA:00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…

Direct

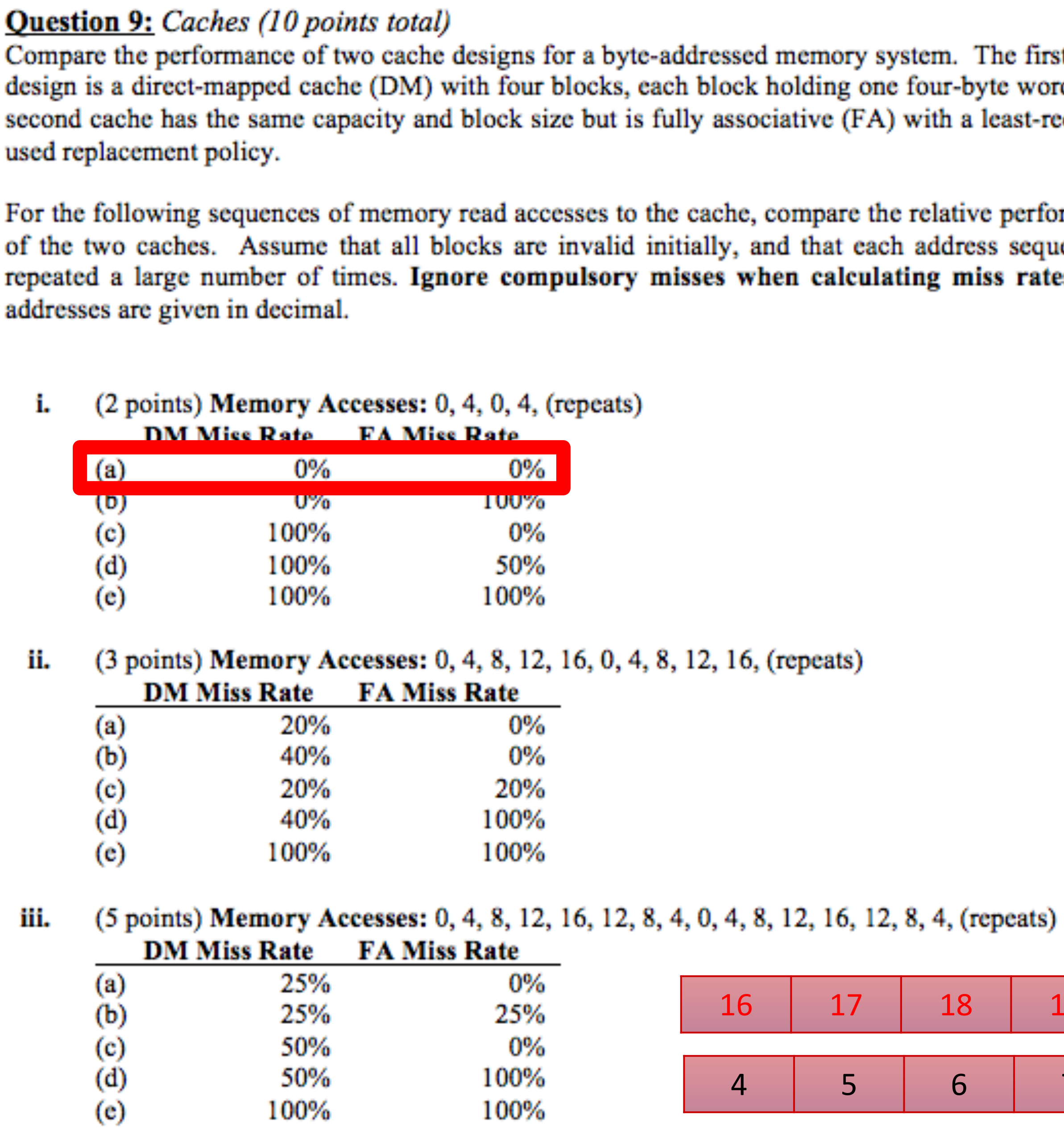
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

FA:

0	1	2	3
4	5	6	7

8	9	10	11
12	13	14	15

Example Problem (Fa 2012)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
(d)	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
(b)	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

ii) Memory Accesses:
D: 00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…
M, M, M, M, M, M, H, H, H, M, …
FA:00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…

Direct

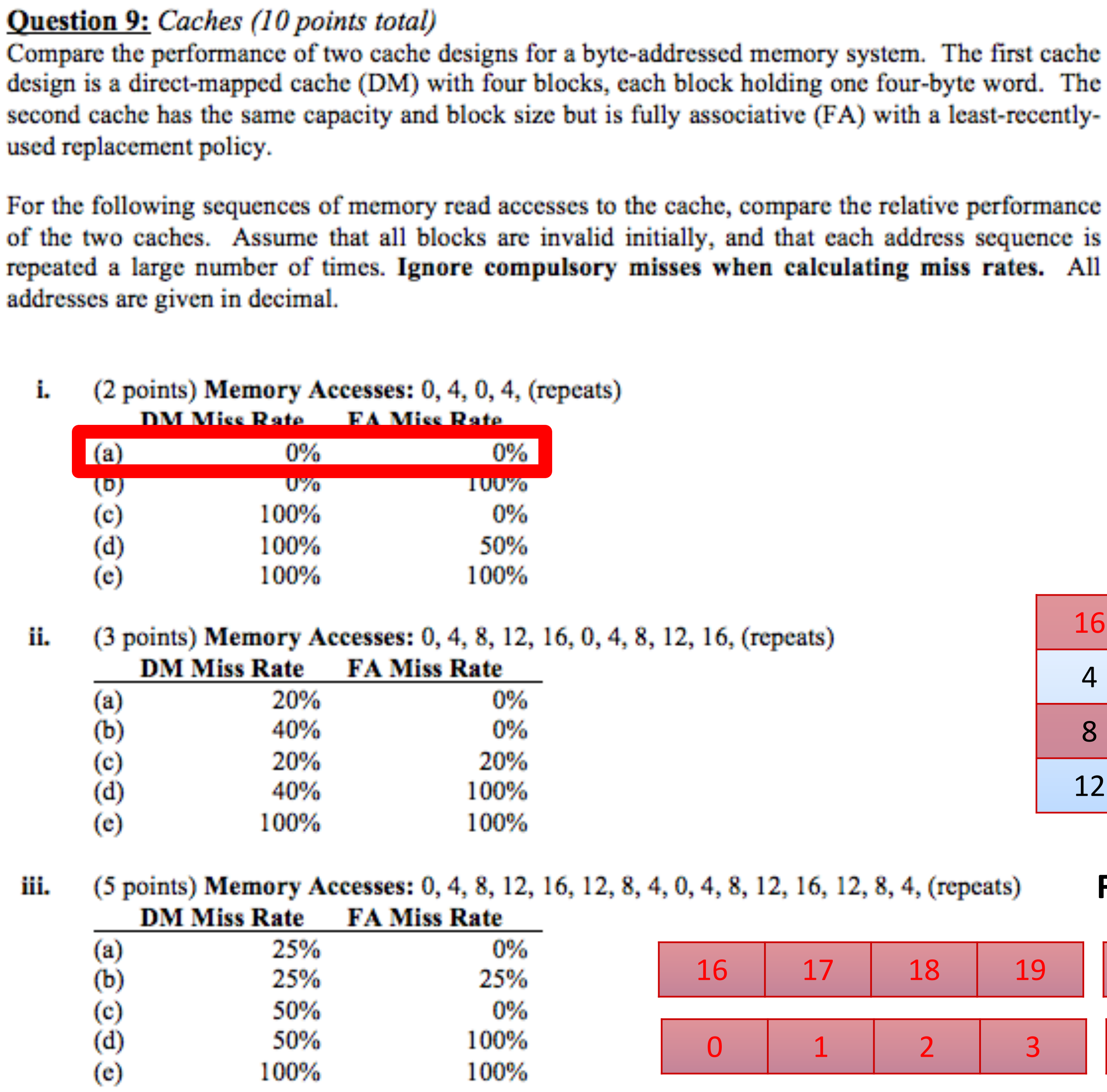
16	17	18	19
4	5	6	7
8	9	10	11
12	13	14	15

FA:

16	17	18	19
4	5	6	7

8	9	10	11
12	13	14	15

Example Problem (Fa 2012)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
(d)	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
(b)	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

ii) Memory Accesses:
D: 00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…
M, M, M, M, M, M, H, H, H, M, …
FA:00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…

Direct

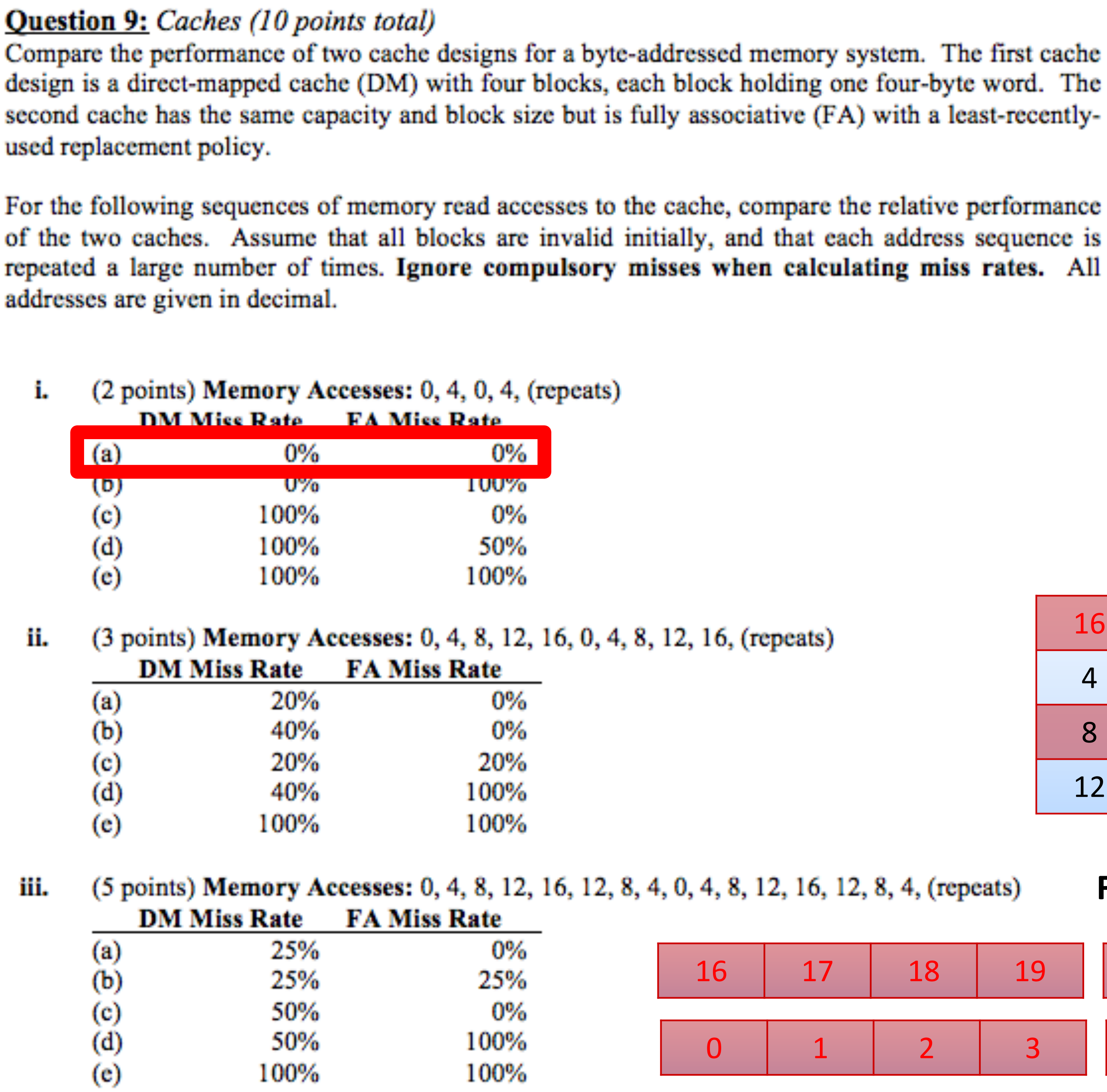
16	17	18	19
4	5	6	7
8	9	10	11
12	13	14	15

FA:

16	17	18	19
0	1	2	3

8	9	10	11
12	13	14	15

Example Problem (Fa 2012)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
(d)	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
(b)	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

ii) Memory Accesses:
D: 00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…
M, M, M, M, M, M, H, H, H, M, …
FA:00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…

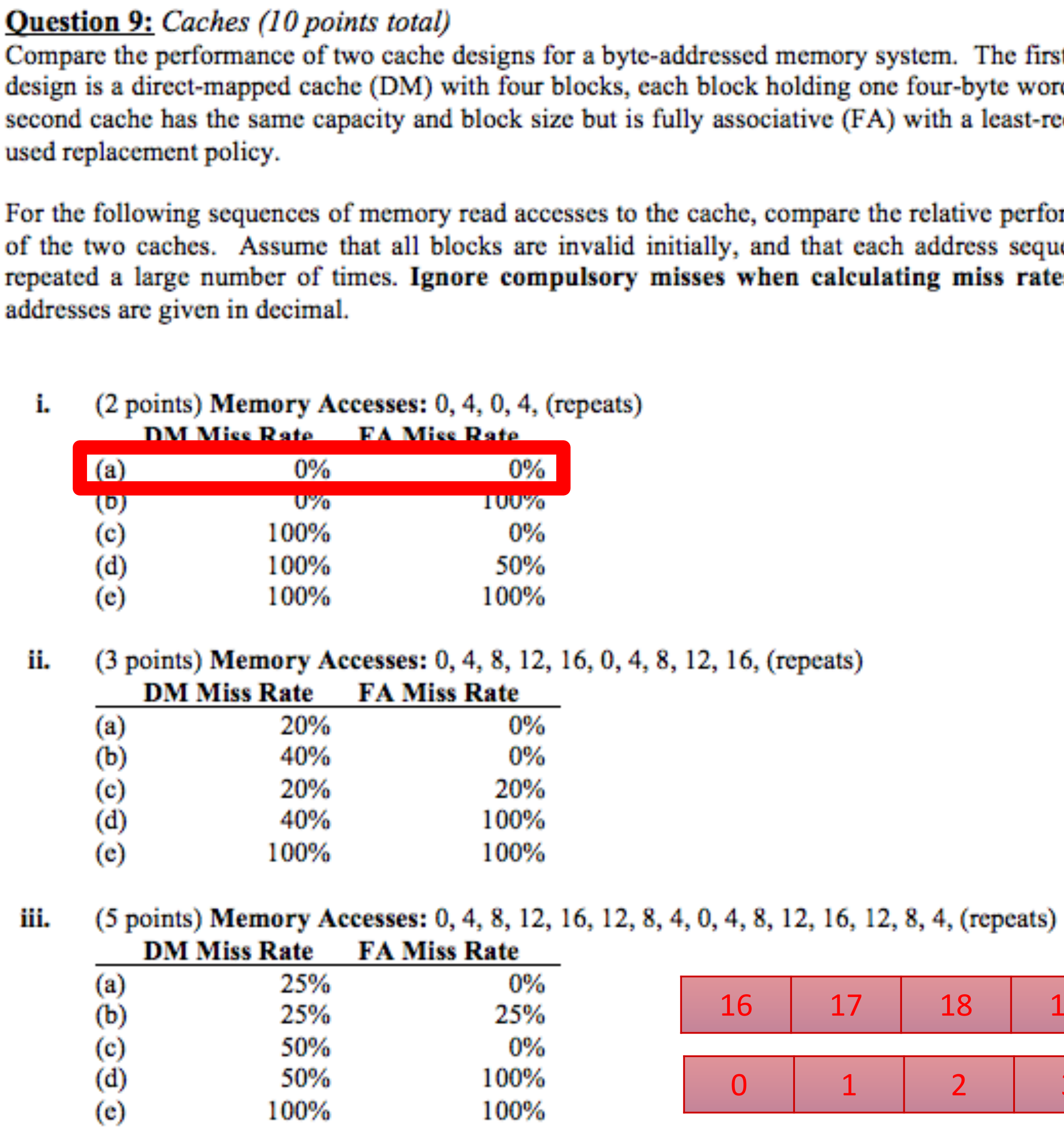
Direct

16	17	18	19
4	5	6	7
8	9	10	11
12	13	14	15

FA:

16	17	18	19	4	5	6	7
0	1	2	3	12	13	14	15

Example Problem (Fa 2012)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
(d)	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
(b)	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

ii) Memory Accesses:
D: 00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…
M, M, M, M, M, M, H, H, H, M, …
FA:00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…

Direct

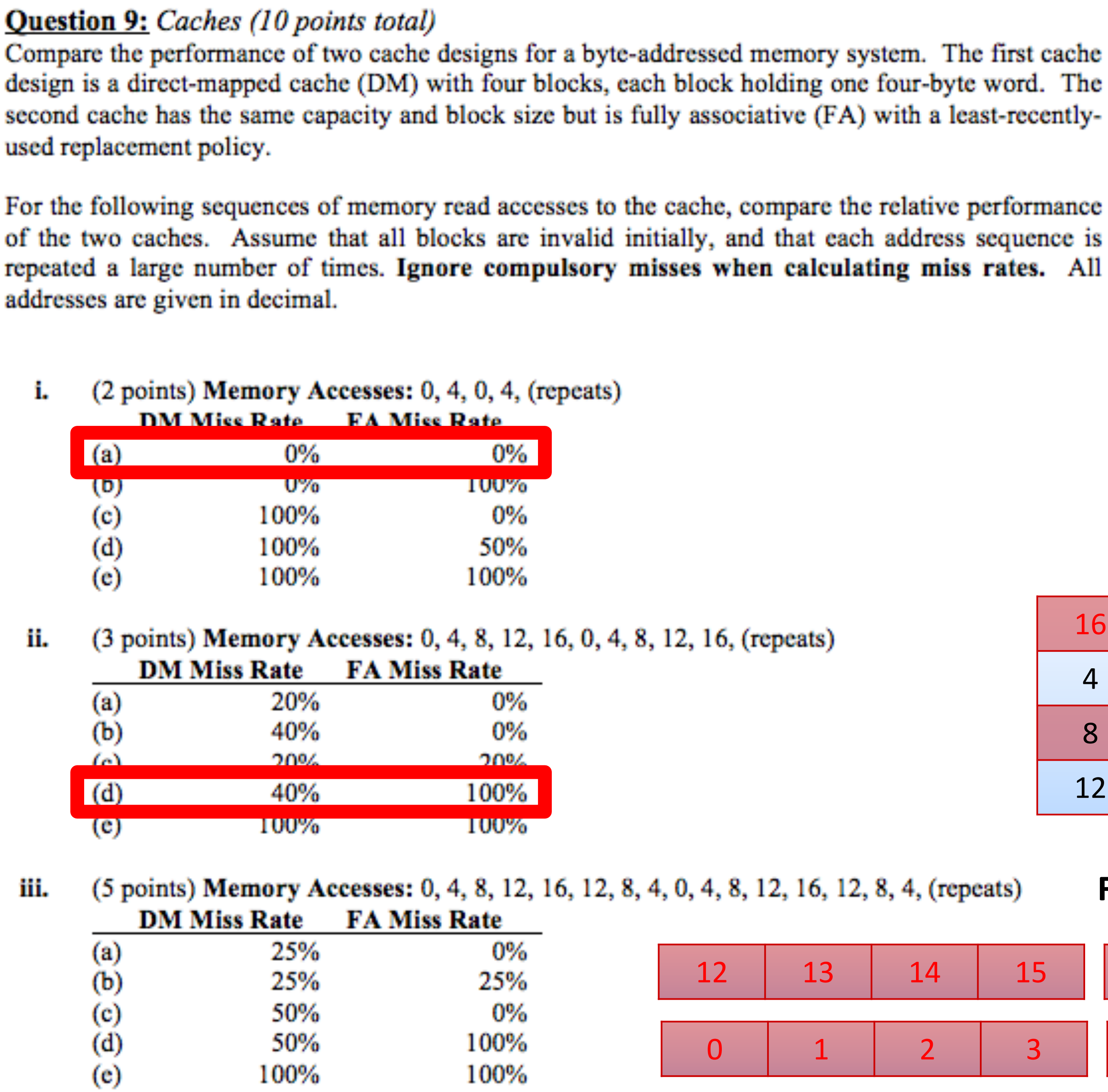
16	17	18	19
4	5	6	7
8	9	10	11
12	13	14	15

FA:

16	17	18	19
0	1	2	3

4	5	6	7
8	9	10	11

Example Problem (Fa 2012)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
[✓ (a)]	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
[✓ (d)]	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
(b)	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

ii) Memory Accesses:
D: 00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…
M, M, M, M, M, M, H, H, H, M, …
FA:00000, 00100, 01000, 01100, 10000, 00000, 00100, 01000, 01100, 10000,…
M, M, M, M, M, M, M, M, M, M, …

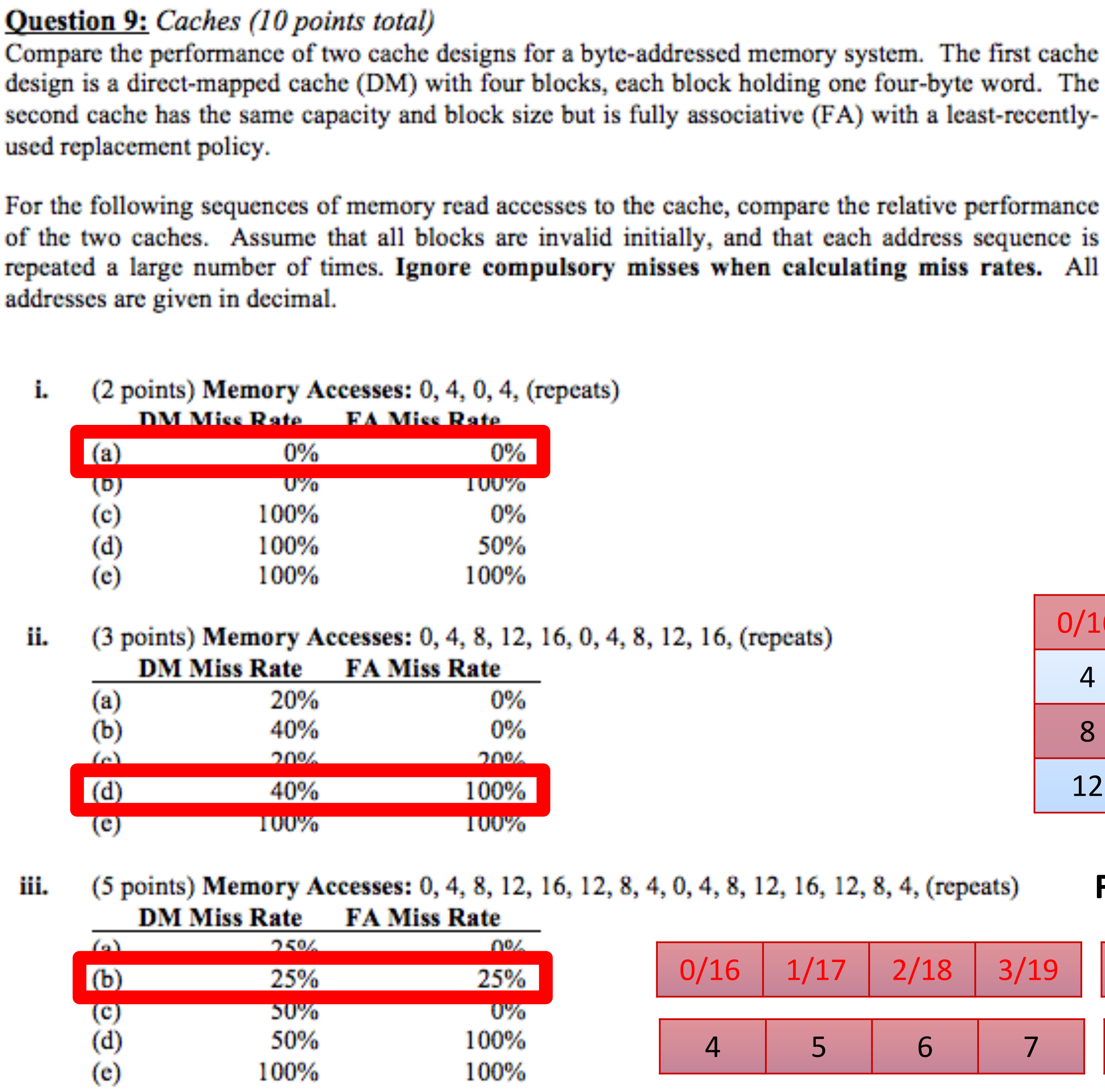
Direct

16	17	18	19
4	5	6	7
8	9	10	11
12	13	14	15

FA:

12	13	14	15	4	5	6	7
0	1	2	3	8	9	10	11

Example Problem (Fa 2012)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
[✓ (a)]	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
[✓ (d)]	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
[✓ (b)]	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

iii) Same as before, except making access backwards as well as forward.

DA: M, M, M, M, M, H, H, H, M, H, H, H, M, H, H, H, …

FA: M, M, M, M, M, H, H, H, M, H, H, H, M, H, H, H, …

Direct

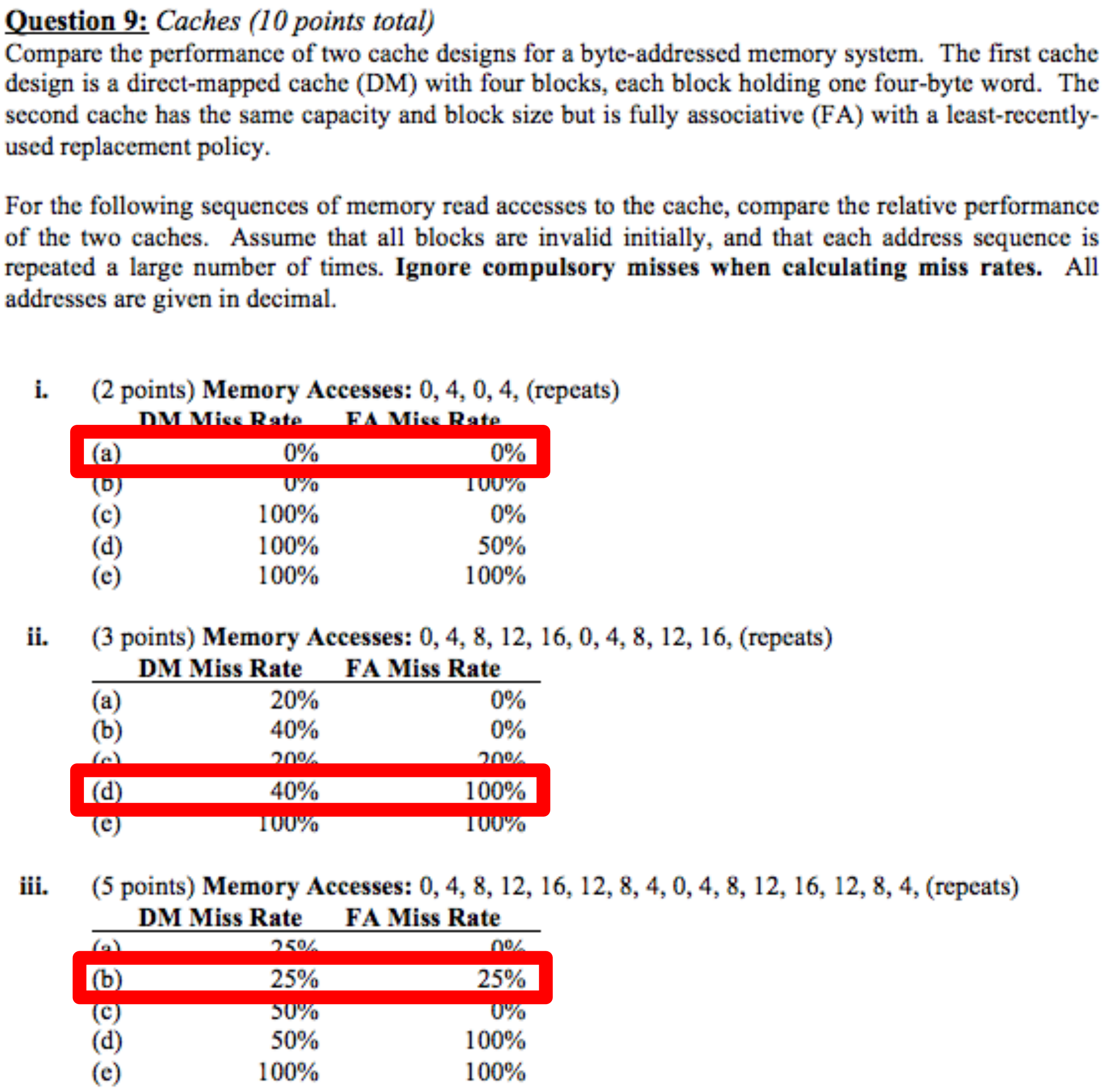
0/16	1/17	2/18	3/19
4	5	6	7
8	9	10	11
12	13	14	15

FA:

0/16	1/17	2/18	3/19
4	5	6	7

8	9	10	11
12	13	14	15

Example Problem (Addition)



Question 9: *Caches (10 points total)*
Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
[✓ (a)]	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
[✓ (d)]	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
[✓ (b)]	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

iv) What is/are the most optimal cache(s) for part ii?
Direct Mapped Cache

v) If you were given memory accesses 0, 16, 4, 20, 0, 16, 4, 20, what is/are the most optimal cache(s)?
2-Way Set Associative Cache
Fully Associative Cache

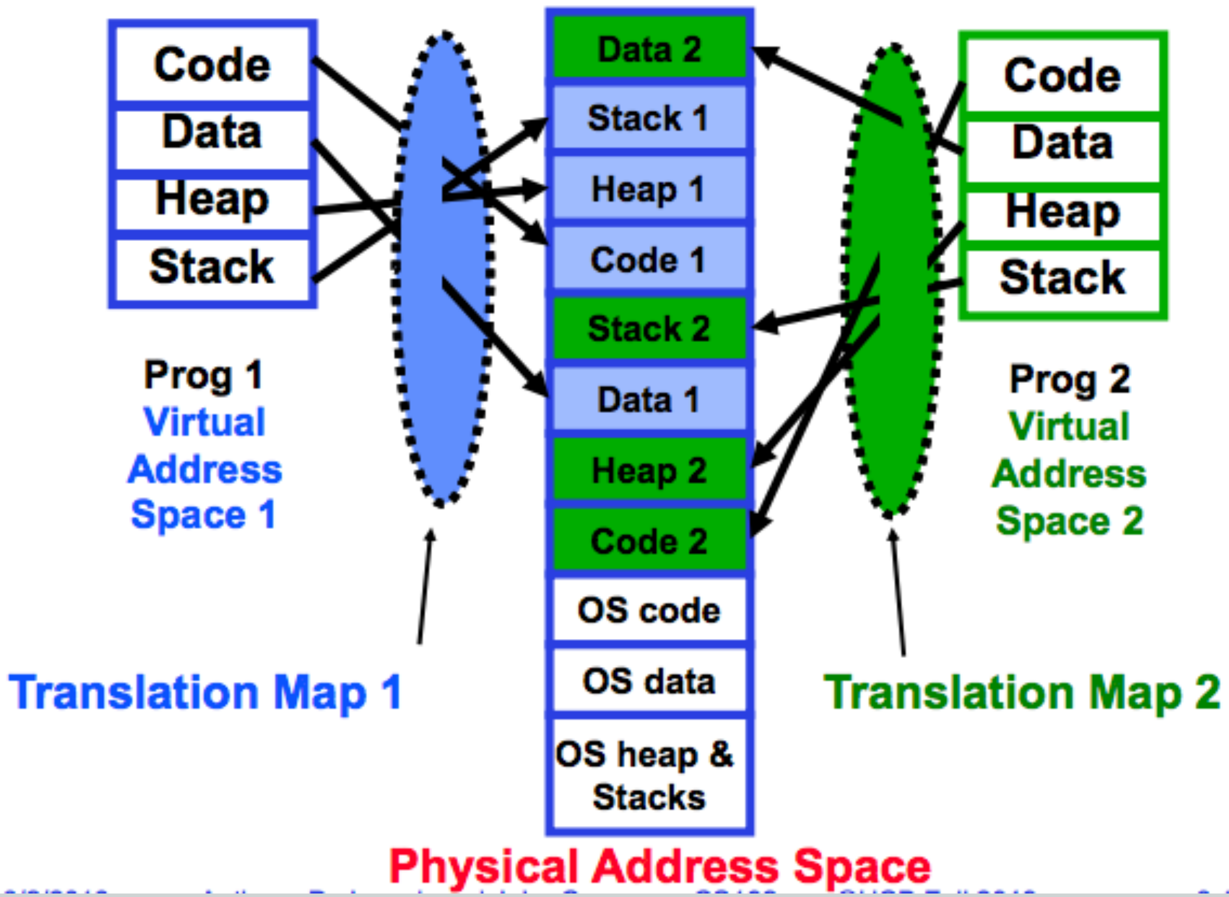
vi) If you were given memory accesses 0, 4, 16, 20, 0, 8, 0, 4, 16, 20, 0, 8 what is/are the most optimal cache(s)?
2-Way Set Associative Cache

vii) How is the most optimal cache for a given computer determined?
Artificial datasets and careful testing. Caches are VERY purpose dependent!

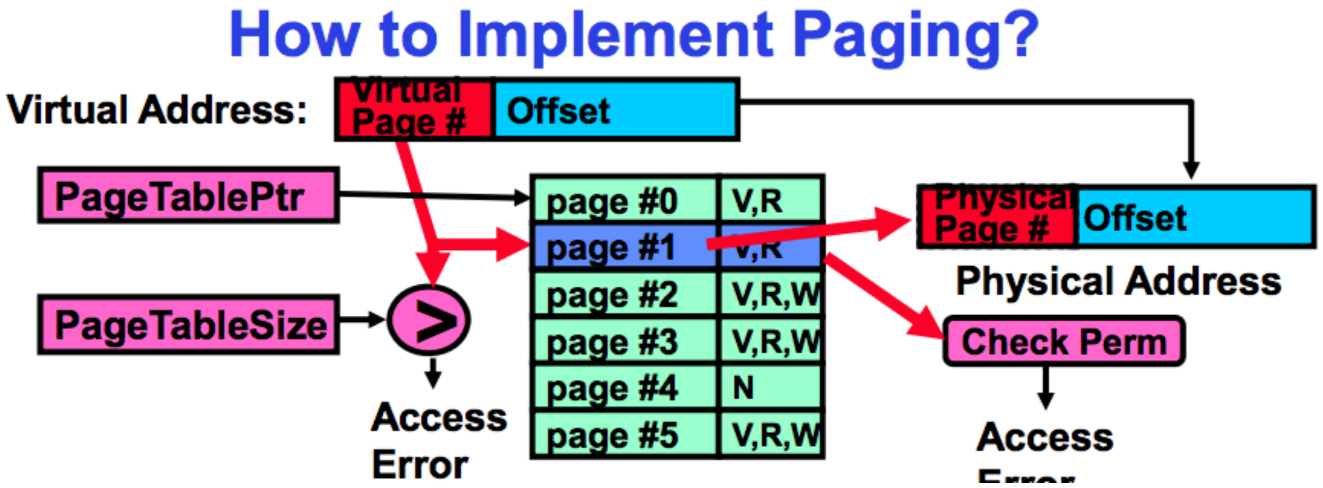
Virtual Memory

A practice question

Brief Overview



Brief Overview



Virtual address (VA): What your program uses

Virtual Page Number	Page Offset

Physical address (PA): What actually determines where in memory to go

Physical Page Number	Page Offset

VPN bits = $\log_2$(VA size / page size)
PPN bits = $\log_2$(PA size / page size)
Page offset = $\log_2$(page size)

Bits per row of PT: PPN bits + valid + dirty + R + W

Tackling a VM question

Spring 2013 #F2:

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

Tackling a VM question

`Spring 2013 #F2:`

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

→ *Before* we start, let's determine the bit-breakdown for virtual and physical addresses.

Tackling a VM question

Spring 2013 #F2:

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

Virtual Addresses:

-offset bits = $\log_2$(size of page in bytes) = $\log_2(2^{20})$ = 20 bits (just like cache offset!)

-VPN bits = $\log_2$(size of virtual memory / size of page in bytes)

= $\log_2(2^{32}/2^{20})$ = 12 bits (just like cache index!)

Tackling a VM question

Spring 2013 #F2:

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

Virtual Addresses:

-offset bits = $\log_2$(size of page in bytes) = $\log_2(2^{20})$ = 20 bits (just like cache offset!)

-VPN bits = $\log_2$(size of virtual memory / size of page in bytes)

= $\log_2(2^{32}/2^{20})$ = 12 bits (just like cache index!)

Physical Addresses:

-offset bits = always the same as virtual pages! $\rightarrow$ 20 bits (just like cache offset!)

-PPN bits = $\log_2$(size of physical memory / size of page in bytes)

= $\log_2(2^{29}/2^{20})$ = 9 bits (just like cache index!)

Tackling a VM question

Spring 2013 #F2:

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

trick to note: VPN + Offset bits = virtual address bits, same for physical memory...

Virtual Addresses:

-offset bits = $\log_2$(size of page in bytes) = $\log_2(2^{20})$ = 20 bits (just like cache offset!)

-VPN bits = $\log_2$(size of virtual memory / size of page in bytes)

= $\log_2(2^{32}/2^{20})$ = 12 bits (just like cache index!)

Physical Addresses:

-offset bits = always the same as virtual pages! $\rightarrow$ 20 bits (just like cache offset!)

-PPN bits = $\log_2$(size of physical memory / size of page in bytes)

= $\log_2(2^{29}/2^{20})$ = 9 bits (just like cache index!)

Tackling a VM question

Spring 2013 #F2:

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

alright, now let's dig into it:

a) How many entries does a page table contain?

Tackling a VM question

Spring 2013 #F2:

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

alright, now let's dig into it:

a) How many entries does a page table contain?

- **we already did this!** Remember, the number of VPN bits tell you how many virtual page translations we can store in the page table at any given time -- just like the index bits of a cache.

Therefore, 12 VPN bits → **2^{12} (virtual) page entries!**
(remember, page table is from VPN to PPN, so we look at VPN bits)

Tackling a VM question

Spring 2013 #F2:

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

b) How wide is the page table base register?

Tackling a VM question

Spring 2013 #F2:

For the following questions, assume the following:

- 32-bit virtual addresses
- 1 MiB pages
- 512 MiB of physical memory with LRU page replacement policy
- Fully associative TLB with 32 entries and an LRU replacement policy

b) How wide is the page table base register?

- **first, remember what the page table base register is:** the page table must live in physical memory somewhere, so the page table base register tells you where to find it!
    - Since the page table lives in *physical memory*, we must have a *physical address* to find it. Therefore, our page table base register must hold a physical address, which is **29 bits wide.**

Tackling a VM question

```
Spring 2013 #F2:
int histogram[MAX_SCORE];
void update_hist(int *scores, int num_scores) {
   for (int i = 0; i < num_scores; i++)
      histogram[scores[i]] += 1;
}
```

Assume that only the code and the two arrays take up memory, ALL of code fits in 1 page, the arrays are page-aligned (start on page boundary), and this is the only process running

c) If update_hist were called with num_scores = 10, how many page faults can occur in the worst-case scenario?

Tackling a VM question

Spring 2013 #F2:

```
int histogram[MAX_SCORE];
void update_hist(int *scores, int num_scores) {
    for (int i = 0; i < num_scores; i++)
        histogram[scores[i]] += 1;
}
```

Assume that only the code and the two arrays take up memory, ALL of code fits in 1 page, the arrays are page-aligned (start on page boundary), and this is the only process running

c) If update_hist were called with num_scores = 10, how many page faults can occur in the worst-case scenario?

some tips:

- -remember the sequential memory layout of arrays
- -don't forget to include any code page faults
- -this is the *worst* case. Be evil!

Tackling a VM question

Spring 2013 #F2:

```
int histogram[MAX_SCORE];
void update_hist(int *scores, int num_scores) {
    for (int i = 0; i < num_scores; i++)
        histogram[scores[i]] += 1;
}
```

Assume that only the code and the two arrays take up memory, ALL of code fits in 1 page, the arrays are page-aligned (start on page boundary), and this is the only process running

c) If update_hist were called with num_scores = 10, how many page faults can occur in the worst-case scenario?

page faults: 11

code -- **0:** In this case we're okay because we have ALL code in the page table/TLB, including the code that called this function

scores -- **1:** it's an array of ints, so with 10 ints that's only 40 B which << 1 page

histogram -- **10:** in the spirit of being evil for the worst case, what if each entry of scores caused histogram to skip a page? We'd page fault all 10 accesses :(

Tackling a VM question

Spring 2013 #F2:

```
int histogram[MAX_SCORE];
void update_hist(int *scores, int num_scores) {
    for (int i = 0; i < num_scores; i++)
        histogram[scores[i]] += 1;
}
```

Assume that only the code and the two arrays take up memory, ALL of code fits in 1 page, the arrays are page-aligned (start on page boundary), and this is the only process running

d) In the best-case scenario, how many iterations of the loop can occur before a TLB miss? You can leave your answer as a product of two numbers.

some tips:

-this is the *best* case. Be forgiving!

-think about the size of your TLB, and what memory accesses we have! (code, score, histogram)

Tackling a VM question

Spring 2013 #F2:

```
int histogram[MAX_SCORE];
void update_hist(int *scores, int num_scores) {
    for (int i = 0; i < num_scores; i++)
        histogram[scores[i]] += 1;
}
```

d) In the best-case scenario, how many iterations of the loop can occur before a TLB miss? You can leave your answer as a product of two numbers.

iterations: $30*2^{18}$

- -firstly, your TLB has 32 entries, one per page.
  - -you need at least one for code, scores, and histogram.
- -now, continuing being nice, what if score[i] was always the same number?
  - -then we'd only need one page for histogram (and only one page for code)
- -this leaves 30 pages to iterate through the score array!
- -each page can hold $(2^{20} / 2^{2}) = 2^{18}$ ints, so we can iterate for **$30*2^{18}$ i values**

Tackling a VM question

Spring 2013 #F2:

```
int histogram[MAX_SCORE];
void update_hist(int *scores, int num_scores) {
    for (int i = 0; i < num_scores; i++)
        histogram[scores[i]] += 1;
}
```

e) For a particular data set, you know the scores are clustered around fifty different values, but you still observe a high number of TLB misses during update_hist. What pre-processing step could help reduce the number of TLB misses?

-Think about when we get a TLB miss.

-If scores are clustered around 50 values, that means we have only ~50 addresses for histogram that we're interested in.

-Remember that the TLB is a cache -- what kinds of locality do we optimize in a cache?

Tackling a VM question

Spring 2013 #F2:

```
int histogram[MAX_SCORE];
void update_hist(int *scores, int num_scores) {
    for (int i = 0; i < num_scores; i++)
        histogram[scores[i]] += 1;
}
```

e) For a particular data set, you know the scores are clustered around fifty different values, but you still observe a high number of TLB misses during update_hist. What pre-processing step could help reduce the number of TLB misses?

sort the values of scores beforehand

Parallelism

Parallelism: across many machines

Big Idea: instead of using expensive super-computers, use 10,000 to 100,000 cheaper servers + networks

Quick facts:

- Servers cost most $$$ (replacement every 3 years)!
- Building/cooling can make these very inefficient!
- A lot of servers sit idle (most servers are at 10-50% load) and waste energy
  - Goal should be

    Energy-Proportionality:

    % peak load = % peak energy

Parallelism: across many machines

PUE:

"Power Usage Effectiveness"

PUE = Total Building Power/IT Power

- Can never be less than 1. Why?

Parallelism: across many machines

PUE:

"Power Usage Effectiveness"

PUE = Total Building Power/IT Power

- Can never be less than 1. Why?
  - IT Power is included in Total Building Power
- Does it measure the effectiveness of server or networking equipment?

Parallelism: across many machines

PUE:

"Power Usage Effectiveness"

PUE = Total Building Power/IT Power

- Can never be less than 1. Why?
  - IT Power is included in Total Building Power
- Does it measure the effectiveness of server or networking equipment?
  - No; instead, PUE focuses on *power* efficiency

Parallelism: across many machines

PUE:

"Power Usage Effectiveness"

PUE = Total Building Power/IT Power

- Can never be less than 1. Why?
  - IT Power is included in Total Building Power
- Does it measure the effectiveness of server or networking equipment?
  - No; instead, PUE focuses on *power* efficiency
- What generally dominates PUE for a WSC?

Parallelism: across many machines

PUE:

"Power Usage Effectiveness"

PUE = Total Building Power/IT Power

- Can never be less than 1. Why?
  - IT Power is included in Total Building Power
- Does it measure the effectiveness of server or networking equipment?
  - No; instead, PUE focuses on *power* efficiency
- What generally dominates PUE for a WSC?
  - Chillers (cooling), followed by IT equipment

Parallelism: across many machines

Sample PUE questions:

Google has 1,000,000 servers in a WSC, and uses 1 MW to power them, and 0.5 MW for cooling and other items in the WSC.

- What is the PUE?
- Say Google reduced its PUE for a given datacenter from 1.7 to 1.2 -- did Google’s server equipment get more powerful? Did Google’s server equipment consume less power?

Parallelism: across many machines

Sample PUE questions:

Which is better? (Assume less $ = better)

WSC 1:

- IT-equip. power:
  - 1000 kW avg
- Non-IT power:
  - 1000 kW avg
- What is the PUE?
  - 2

WSC 2:

- IT-equip. power:
  - 1600 kW avg
- Non-IT power:
  - 600 kW avg
- What is the PUE?
  - 1.375

(from Sagar Karandikar’s discussion slides)

Parallelism: across many machines

Sample PUE questions:

Which is better? (Assume less $ = better)

WSC 1:

- IT-equip. power:
  - 1000 kW avg
- Non-IT power:
  - 1000 kW avg
- What is the PUE?
  - 2

WSC 2:

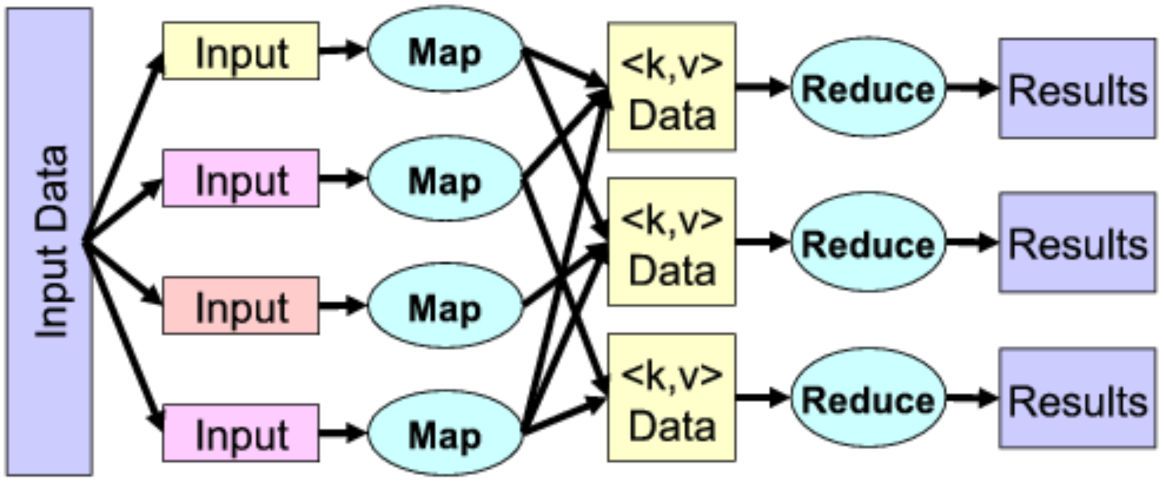
- IT-equip. power:
  - 1600 kW avg
- Non-IT power:
  - 600 kW avg
- What is the PUE?
  - 1.375

Even though it has a lower PUE, WSC 1 is cheaper and **better**
(if all other things, like compute cycles, are held equal)

(from Sagar Karandikar's discussion slides)

Parallelism: across many machines

MapReduce:



- Map, Shuffle, Reduce stages
- Can have multiple Map or Reduce stages (like in the project!)
- Typically have a master node for managing machines, coping with failures
- Perfect for sharding **large** quantities of data with multiple machines

Parallelism: across many machines

```
# Word count in Spark -- usually ~100 lines in Hadoop!

file = spark.textFile("hdfs://...")
counts = file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
```

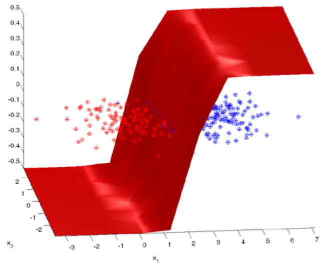
-Spark abstracts away the map() and reduce() functions into higher level functions operating on **Resilient Distributed Datasets** (RDDs)

Parallelism: across many machines

Let's use Spark to make a logistic regression (machine learning) classifier!

To make a logistic regression classifier,

we must determine the value of a vector: `w`



`w` will specify a separating plane, separating our two classes (perhaps spam v. non-spam, etc.). The details of how we proceed using `w` are not too important -- for this question we just wish to compute it.

To compute `w`, it is in iterative process: first you compute a weighted term for each point, and add them together to get a gradient. Then, you subtract this gradient from `w`, and repeat!

Parallelism: across many machines

Your job: compute `w` using Spark!

The algorithm: for ITERATIONS, compute the weighted term for each point, then sum them all together. Subtract this from `w` and repeat.

Here's some starter code, you can assume that you have a pre-existing `w` as well as a function to compute the weighted term for a specific point:

pseudocode:

```
# current separating plane
for i in range(ITERATIONS):
     gradient = 0
     for p in points:
          gradient += computeWeightedTerm(p)
     w -= gradient

print "Final separating plane: %s" % w
```

Parallelism: across many machines

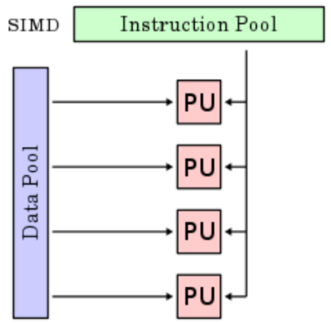
```
# current separating plane
for i in range(ITERATIONS):
    gradient = points.map(computeWeightedTerm)

                .reduce(lambda a, b: a + b)

    w -= gradient


print "Final separating plane: %s" % w
```

Look how short the code is! There are other nice examples on: spark.
apache.org

Parallelism: on a single machine

SIMD: Intel SSE

- SIMD: Single Instruction Multiple Datastream
    - Flynn's taxonomy
- Intel SSE is an implementation of SIMD, defining its own 128 bit registers
    - These registers can be split into 4 ints, 4 floats, 2 doubles, etc.
    - You can tell which type it is from the instruction name!
        - `_mm_add_epi32 vs. _mm_add_epi64`
- Remember about remainder cases!



Parallelism: on a single machine

What’s wrong with this code?

This code should multiply the first n elements of 32-bit integer array a[ ]

You should be able to find a couple of bugs.

```
int mul_array(const int a[], const int n) {
      __m128i vmul = _mm_set1_epi32(1); // initialise vector of four partial 32 bit products
      int i;
      for (i = 0; i < n; i += 4) {
            __m128i v = _mm_loadu_si128(&a[i]); // load vector of 4 x 32 bit values
            vmul = _mm_mul_epi64(vmul, v); // accumulate to 32 bit partial product vector
      }
      int prod[4] = {0, 0, 0, 0};
      _mm_storeu_si128(prod, vmul);
      return prod[0] * prod[1] * prod[2] * prod[3];
}
```

Parallelism: on a single machine

```
int mul_array(const int a[], const int n) {
      __m128i vmul = _mm_set1_epi32(1); // initialise vector of four partial 32 bit sums
      int i;
      for (i = 0; i < n / 4 * 4; i += 4) {        // what if n % 4 != 0?
            __m128i v = _mm_loadu_si128(&a[i]); // load vector of 4 x 32 bit values
            vmul = _mm_mul_epi32(vmul, v);        // remember to use the right instruction!
      }
      int prod[4] = {0, 0, 0, 0};
      _mm_storeu_si128((__m128i*) prod, vmul);    // a common bug, remember these need _m128i’s
      for (int i = n / 4 * 4; i < n; i++) {      // the tail case!
            prod[0] *= a[i];
      }

      return prod[0] * prod[1] * prod[2] * prod[3];
}
```

Parallelism: on a single machine

openMP:

- Use the power of multi-threading!
- Works via the use of directives:
  - `#pragma omp parallel`
  - `#pragma omp parallel for`
  - `#pragma omp critical`
  - `etc...`

- Each thread is independently executed:
  - But variables are *shared* by default
    - with the exception of loop indices!

Parallelism: on a single machine

openMP:

- Data races
  - Different threads access the **same** location (stack or heap) one after another, and at least one is performing a write
  - This is why we have `#pragma omp critical`

Parallelism: on a single machine

```
#pragma omp parallel for
for (int x = 0; x < len; x++) {
     *A = x;
     A++;
}

#pragma omp parallel
{
     for (int x = 0; x < len; x++) {
          *(A+x) = x;
     }
}

#pragma omp parallel
{
     for(int x = omp_get_thread_num(); x < len; x += omp_get_num_threads()) {
          A[x] = x;
     }
}
```

Spring 2013, #4:

- Are any of these implementations correct?
- Which implementation is fastest?
- How does each compare to a serial implementation?

Parallelism: on a single machine

```
#pragma omp parallel for
for (int x = 0; x < len; x++) {
     *A = x; // data race!
     A++;
}

#pragma omp parallel
{
     for (int x = 0; x < len; x++) {
          *(A+x) = x;
     }
}

#pragma omp parallel
{
     for(int x = omp_get_thread_num(); x < len; x += omp_get_num_threads()) {
          A[x] = x;
     }
}
```

Spring 2013, #4:

- Are any of these implementations correct?
- Which implementation is fastest?
- How does each compare to a serial implementation?

Parallelism: on a single machine

```
#pragma omp parallel for
for (int x = 0; x < len; x++) {
     *A = x; // data race!
     A++;
}

#pragma omp parallel // repeated work
{
     for (int x = 0; x < len; x++) {
          *(A+x) = x;
     }
}

#pragma omp parallel
{
     for(int x = omp_get_thread_num(); x < len; x += omp_get_num_threads()) {
          A[x] = x;
     }
}
```

Spring 2013, #4:

- Are any of these implementations correct?
- Which implementation is fastest?
- How does each compare to a serial implementation?

Parallelism: on a single machine

```
#pragma omp parallel for
for (int x = 0; x < len; x++) {
     *A = x; // data race!
     A++;
}

#pragma omp parallel // repeated work
{
     for (int x = 0; x < len; x++) {
          *(A+x) = x;
     }
}

#pragma omp parallel
{
     for(int x = omp_get_thread_num(); x < len; x += omp_get_num_threads()) {
          A[x] = x; // this is just like your lab!  It works, but is it fast?
     }
}
```

Spring 2013, #4:

- Are any of these implementations correct?
- Which implementation is fastest?
- How does each compare to a serial implementation?

Parallelism: on a single machine

Spring 2014 Final #4

```
void outer_product(float* dst, float *x, float *y, size_t n) {
    for (size_t i = 0; i < n; i += 1)
        for (size_t j = 0; j < n; j += 1)
            dst[i*n + j] = x[i] * y[j];
}
```

Consider the unparallelized outer product code, above.

If x and y are two column vectors, $O = xy^T$, such that $O_{ij} = x_i y_j$

Parallelism: on a single machine

Spring 2014 Final #4a

```
void outer_product(float* dst, float *x, float *y, size_t n) {
    ________________________________
    for (size_t i = 0; i < n; i += 1)
        _________________________________
        for (size_t j = 0; j < n; j += 1)
            _______________________________
            dst[i*n + j] = x[i] * y[j];
}
```

Insert openMP directives in the blanks to best parallelize the code!

You can use as little/many of the blanks as you want

Parallelism: on a single machine

Spring 2014 Final #4a

```
void outer_product(float* dst, float *x, float *y, size_t n) {
    #pragma omp parallel for
    for (size_t i = 0; i < n; i += 1)
        // if you inserted here, it wouldn’t help much…
        for (size_t j = 0; j < n; j += 1)
            // no need for a critical section
            dst[i*n + j] = x[i] * y[j];
}
```

-Remember that **for** means that openMP automatically breaks up the threads

-We don’t need to worry about any critical sections because each section is independent! Each entry in memory is only edited by one thread.

Parallelism: on a single machine

Spring 2014 Final #4b

```
void outer_product(float* dst, float *x, float *y, size_t n) {
    for (size_t i = 0; i < n; i += 1)
        for (size_t j = 0; j < n; j += 1)
            __________________________________
            __________________________________
            __________________________________
            _mm_storeu_ps(&dst[ i*n + j ], products);
            j += 3;
}
```

Insert SSE Intrinsics to optimize the code!

You can assume n is a multiple of 4

Parallelism: on a single machine

Spring 2014 Final #4b

```
void outer_product(float* dst, float *x, float *y, size_t n) {
    for (size_t i = 0; i < n; i += 1)
        for (size_t j = 0; j < n; j += 1)
            _________________________________
            _________________________________
            _________________________________
            _mm_storeu_ps(&dst[ i*n + j ], products);
            j += 3;
}
```

Before we start: note how we're writing to memory -- for each storeu we're writing to O_{ij}, O_{ij+1}, O_{ij+2}, O_{ij+3}

Parallelism: on a single machine

Spring 2014 Final #4b

```
void outer_product(float* dst, float *x, float *y, size_t n) {
    for (size_t i = 0; i < n; i += 1)
        for (size_t j = 0; j < n; j += 1)
            __m128 a = _mm_load1_ps(&x[i]);
            _______________________________
            _______________________________
            _mm_storeu_ps(&dst[ i*n + j ], products);
            j += 3;
}
```

Before we start: note how we're writing to memory -- for each storeu we're writing to O_{ij}, O_{ij+1}, O_{ij+2}, O_{ij+3}

Parallelism: on a single machine

Spring 2014 Final #4b

```
void outer_product(float* dst, float *x, float *y, size_t n) {
    for (size_t i = 0; i < n; i += 1)
        for (size_t j = 0; j < n; j += 1)
            __m128 a = _mm_load1_ps(&x[i]);
            __m128 b = _mm_loadu_ps(&y[j]);
            __________________________________
            _mm_storeu_ps(&dst[ i*n + j ], products);
            j += 3;
}
```

Before we start: note how we're writing to memory -- for each storeu we're writing to O_{ij}, O_{ij+1}, O_{ij+2}, O_{ij+3}

Parallelism: on a single machine

Spring 2014 Final #4b

```
void outer_product(float* dst, float *x, float *y, size_t n) {
    for (size_t i = 0; i < n; i += 1)
        for (size_t j = 0; j < n; j += 1)
            __m128 a = _mm_load1_ps(&x[i]);
            __m128 b = _mm_loadu_ps(&y[j]);
            __m128 products = _mm_mul_ps(a, b);
            _mm_storeu_ps(&dst[ i*n + j ], products);
            j += 3;
}
```

Before we start: note how we're writing to memory -- for each storeu we're writing to O_{ij}, O_{ij+1}, O_{ij+2}, O_{ij+3}

Digital Systems

Gates and Delay

There are three main limiting factors to the frequency of a circuit:

- setup time: how much time the input must be stable ***before*** the rising edge
- hold time: how much time the input must be stable ***after*** the rising edge
- clk-to-q delay: how long it takes the output to reflect the input ***after the rising edge***
- Max Delay = CLK-to-Q Delay + Combinational Logic Delay (e.g., adder delay) + Setup Time

Gates and Delay, Example

You’re an intern at a circuit design company, and your first job is to choose the cheapest possible adder for a complicated circuit. The constraints that you’re given are the following:

- assume that higher propagation delay means a cheaper circuit
- the adder feeds directly into a register
- the inputs to the adder come from a clocked register
- the clock speed of the circuit is 4GHz (1/250 ps)
- assume all registers in the circuit are the same
- The delay of the rest of the circuit is 140ps
- register information:
    - setup time: 15ps
    - hold time: 25 ps
    - clock-to-q: 20ps

You have access to adders with any propagation delay that you’d like. What propagation delay do you choose?

Gates and Delay, Example

Max Delay: 250ps

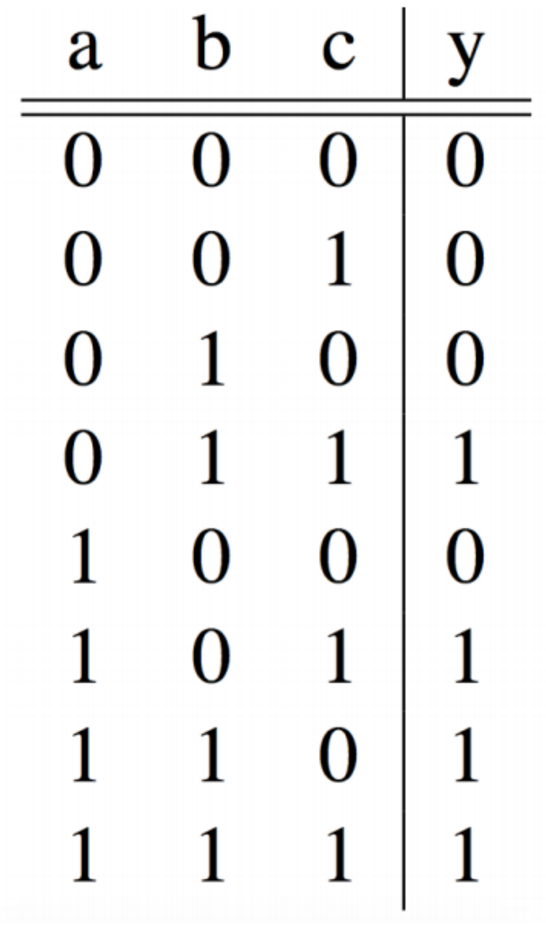
Max Delay = (rest of circuit) + clk-to-q delay + adder delay + setup time

adder delay = 250 - 140 - 20 - 15 = 75ps.

Boolean Logic

Example Truth Table:

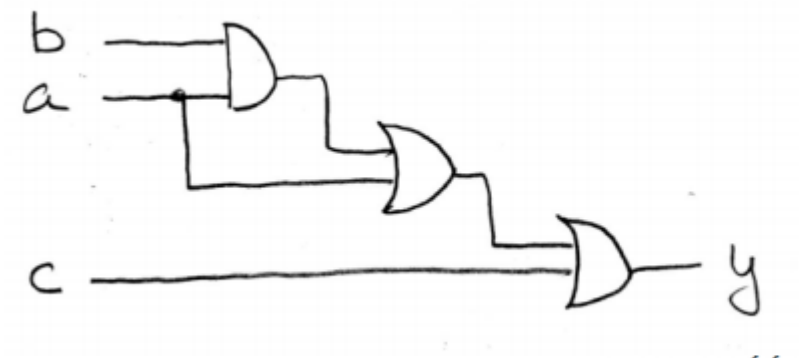
Majority Circuit



a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(taken from Lecture 25 slides)

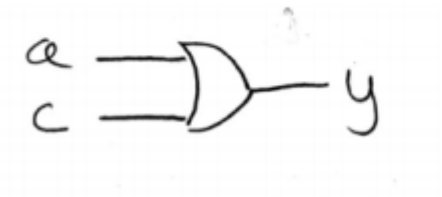
Boolean Algebra



$$y = ((ab) + a) + c$$

$$\downarrow$$

$$\begin{aligned} &= ab + a + c \\ &= a(b+1) + c \\ &= a(1) + c \\ &= a + c \end{aligned}$$



(taken from Lecture 25 slides)

Boolean Algebra

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	complementarity
$x \cdot 0 = 0$	$x + 1 = 1$	laws of 0's and 1's
$x \cdot 1 = x$	$x + 0 = x$	identities
$x \cdot x = x$	$x + x = x$	idempotent law
$x \cdot y = y \cdot x$	$x + y = y + x$	commutativity
$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$	associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	distribution
$xy + x = x$	$(x + y)x = x$	uniting theorem
$\overline{x}y + x = x + y$	$(\overline{x} + y)x = xy$	uniting theorem v.2
$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$	DeMorgan's Law

Boolean Algebra, Example 1

$(A+B)(A+\bar{B})C$

$\overline{A(\bar{B}\bar{C}+BC)}$

Boolean Algebra, Example 1

$$(A+B)(A+\bar{B})C$$

$$(AA+A\bar{B}+AB+B\bar{B})C=(A+A(\bar{B}+B))C=AC$$

$$\overline{A(\bar{B}\bar{C}+BC)}$$

Boolean Algebra, Example 1

$$(A+B)(A+\bar{B})C$$

$$(AA+A\bar{B}+AB+B\bar{B})C=(A+A(\bar{B}+B))C=AC$$

$$\overline{A(\bar{B}\bar{C}+BC)}$$

$$\begin{aligned}
\overline{A(\bar{B}\bar{C}+BC)} &= \bar{A}+\overline{\bar{B}\bar{C}+BC} \\
&= \bar{A}+\overline{\bar{B}\bar{C}}\,\overline{BC} \\
&= \bar{A}+(B+C)(\bar{B}+\bar{C}) \\
&= \bar{A}+B\bar{C}+\bar{B}C
\end{aligned}$$

Boolean Algebra, Example 2

```
¬A(A+B)  +  (B + AA)(A + ¬B)
```

Boolean Algebra, Example 2

$\neg A(A+B) + (B + AA)(A + \neg B)$

$(\neg A)A + (\neg A)B + (B + A)(A + \neg B)$

$(\neg A)A + (\neg A)B + BA + B(\neg B) + AA + A(\neg B)$

$0 + (\neg A)B + AB + 0 + A + A(\neg B)$

$B(A + \neg A) + A(1 + \neg B)$

$A + B$

Finite State Machines

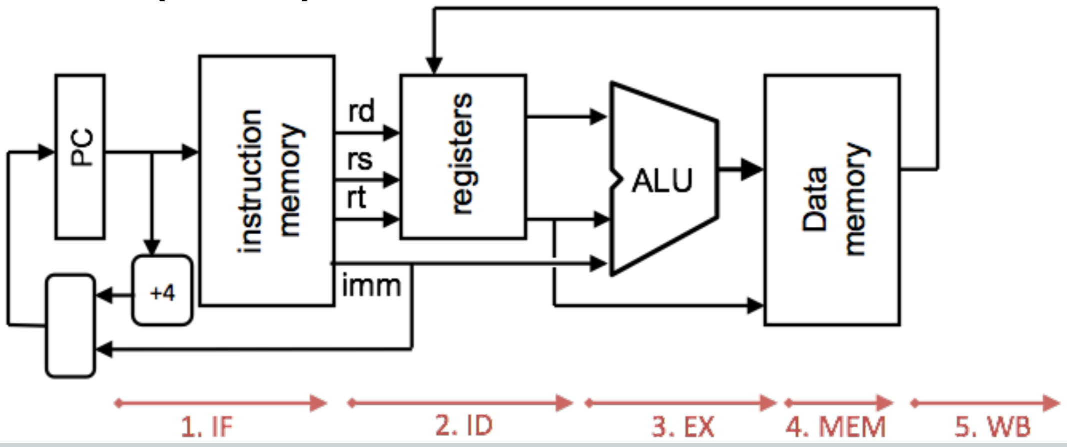
What is an FSM?

- Series of “states” that the system can be in
- Input to current state define the output and the next state of the system
- Can be represented with combinational logic/truth tables

MIPS Datapath

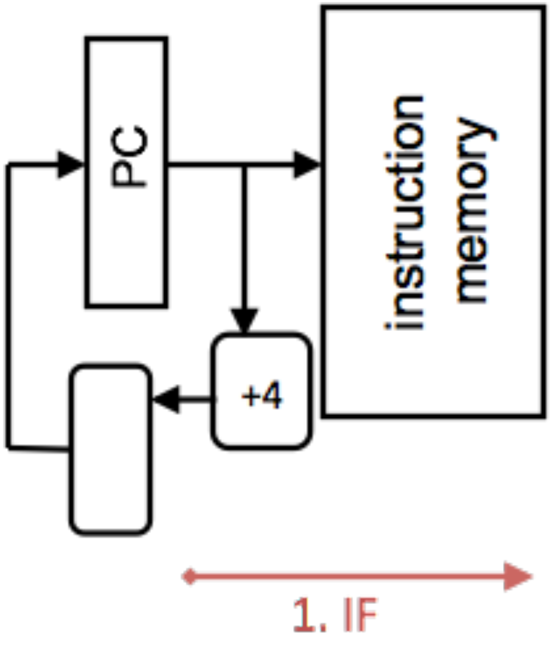
Five stages:

1. I-Fetch (IF)
2. I-Decode (ID)
3. ALU (EX)
4. Memory Access (MEM)
5. Write to Register (WB)



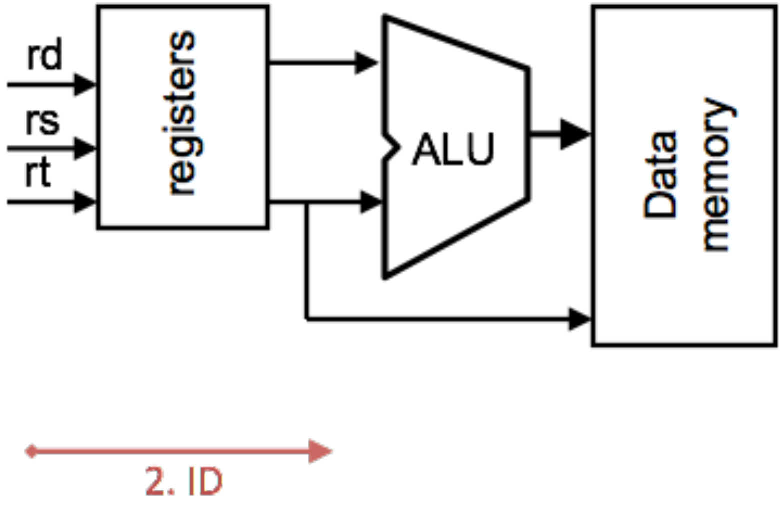
IF

1. Fetch the instruction from memory.
2. Increment PC by 4 (or change to new address in the case of a branch or jump).



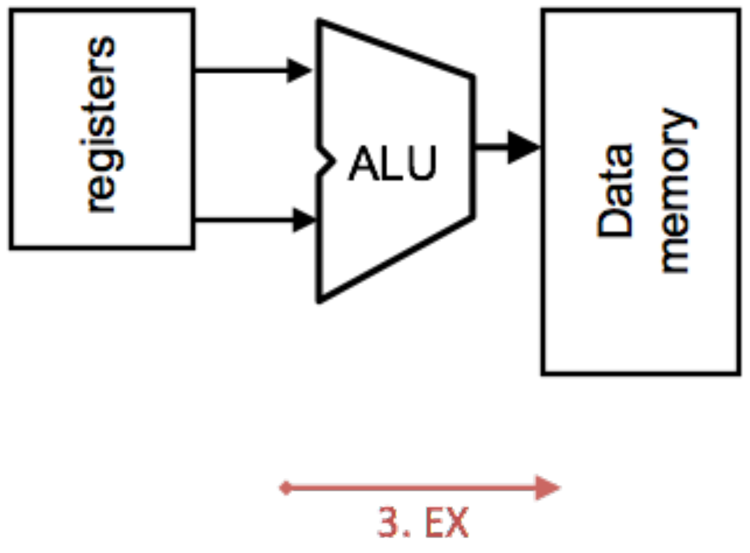
ID

- Decode instruction into necessary fields
    - opcode, maybe function
    - get registers (rs, rt, rd)
    - shamt, address, immediate (varies by instruction)
- All information is decoded but only certain pieces are used.



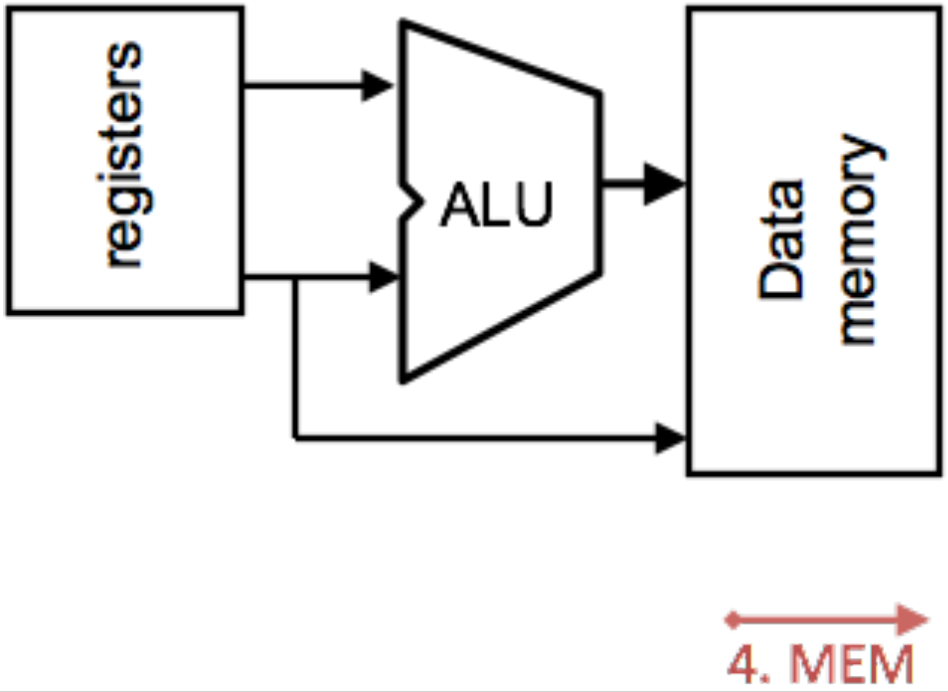
EX

- All calculations done here (adding, shifting, comparing, etc.)
  - For loads and stores, memory address computation is done here



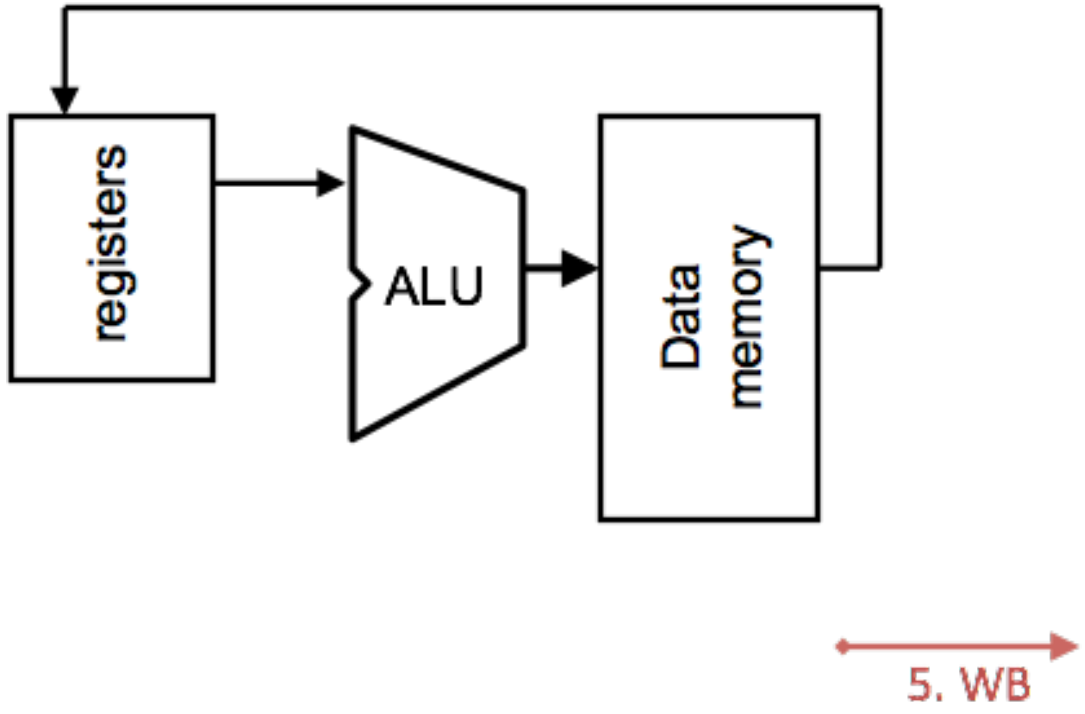
MEM

- For loads and stores, write or read contents to/from memory address
- For all other instructions, idle



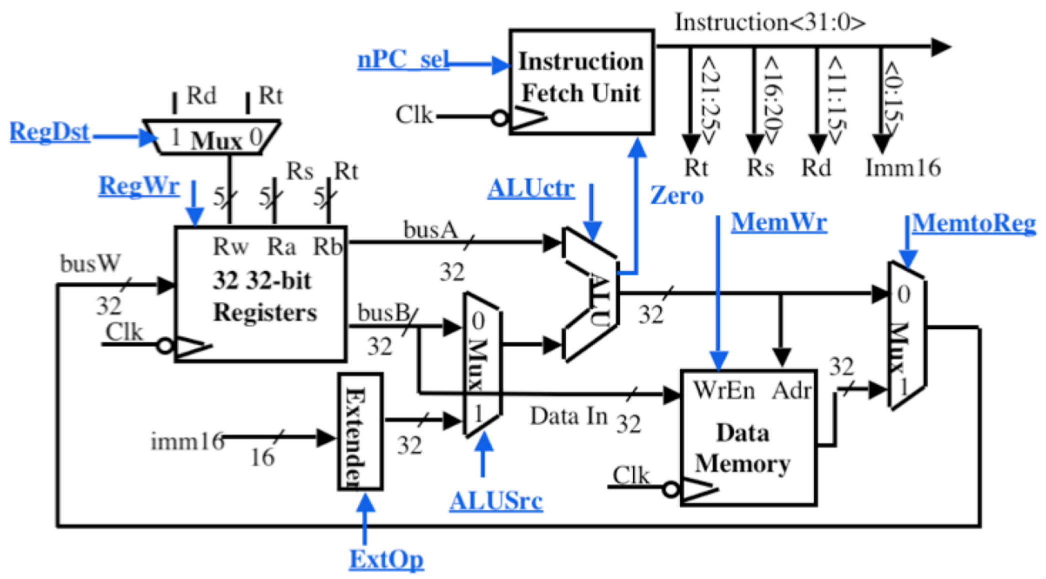
WB

- Write the appropriate value from the computation into the register
  - not used for stores, branches, jumps (excluding jal, jalr)



Control Signals

- `RegDst`: `rt` or `rd`
- `RegWr`: write reg?
- `ExtOp`: sign or zero
- `ALUSrc`: BusB or Imm
- `ALUctr`: choose op
- `MemWr`: write mem?
- `MemtoReg`: ALU or Mem
- `nPC_sel`: +4, br, or j



Single Cycle Datapath: New Instructions

We want to add a new MIPS instruction (we'll call it `addpr` for "add to pointed reg") that is almost identical to `addi` but with a twist. Instead of storing into the `rt` register the sum of the constant and the value of the register specified by `rs`, it stores into the `rt` register the sum of the constant and the value of the register specified by the lowest 5 bits in memory at the address specified by the pointer stored in the `rt` register.

Said another way:

-First get the pointer stored in the `rt` register.

-Follow that pointer to get its value from memory.

-Take the lowest 5 bits of that value, treat is as a register number, and find out what is stored in that register.

-Add that to the immediate, and store it in the `rt` register.

Single Cycle Datapath: New Instructions

The syntax is: `addpr $v0, 5`

Write the RTL:

Single Cycle Datapath: New Instructions

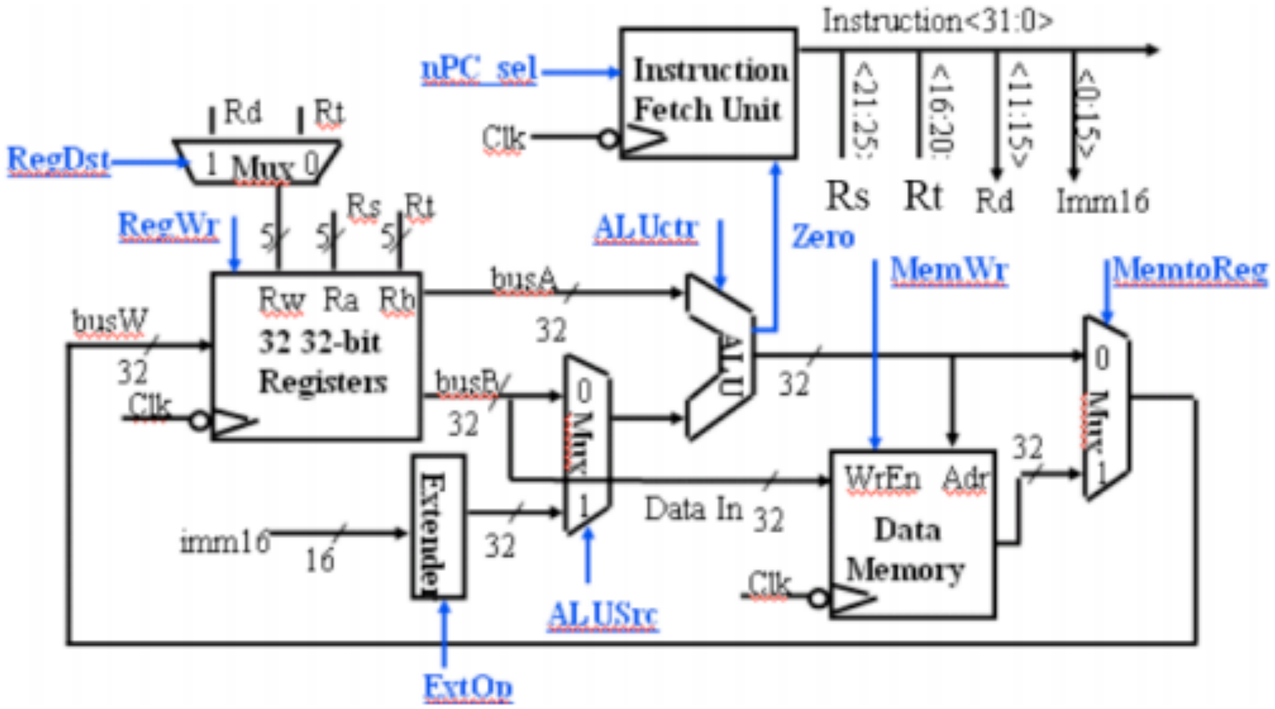
The syntax is: `addpr $v0, 5`

Write the RTL:

```
    R[rt] = R[MEM[R[rt]](4:0)] + SignExtImm;
    PC = PC + 4
```

Single Cycle Datapath: New Instructions

What changes do you need to make to the regular single-cycle datapath?



Single Cycle Datapath New Instructions

1. Add a mux whose output is tied to “`Data Memory Adr`” and whose input is either the `ALU` or `busB R[rt]`, driven by a control line called “`MemAdr`” whose value is either `ALU` or `busB`

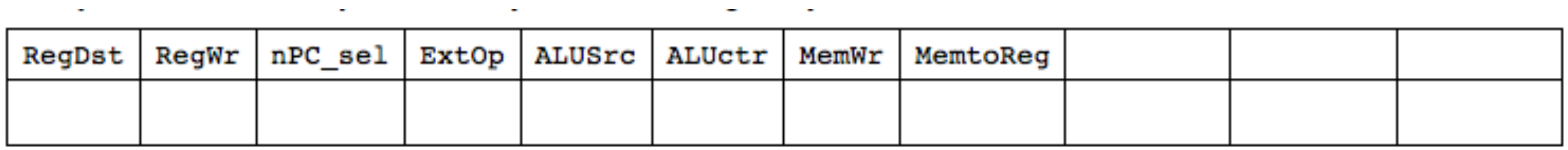
Single Cycle Datapath New Instructions

1. Add a mux whose output is tied to "`Data Memory Adr`" and whose input is either the `ALU` or `busB R[rt]`, driven by a control line called "`MemAdr`" whose value is either `ALU` or `busB`

2. Add a mux whose output is tied to "`Ra`" and whose input is either `Rs` or the lowest 5 bits of "`Data Memory Data Out`", driven by a control line called "`RaSrc`" whose value is either `Rs` or `Mem`

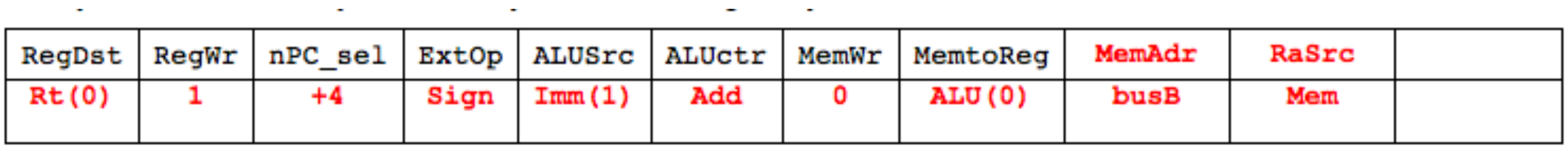
Single Cycle Datapath: New Instructions

What are the values of all the control bits?



RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg			

Single Cycle Datapath: New Instructions



RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg	MemAdr	RaSrc	
Rt(0)	1	+4	Sign	Imm(1)	Add	0	ALU(0)	busB	Mem	

Feedback

We would like your feedback on this review session, so that we can improve for future final review sessions.

(This is our first time doing a CS61C final review!)

If you would like to provide suggestions or constructive criticism, please fill out the paper copy being distributed.

Thank you, and best of luck for your final! :)

APPENDIX

C: Warm-up - Pointers and Arrays

```
int num = 5; //num == ??
int *metaNum = &num; //metaNum == ??
int **metaMetaNum = &metaNum; //???

printf(“%lu”, *metaMetaNum);
???
printf(“%lu”, **metaMetaNum);
???
printf(“%lu”, *num);
???
```

C: Warm-up - Pointers and Arrays

```
int num = 5; //num == 5
int *metaNum = &num; //metaNum == ??
int **metaMetaNum = &metaNum; //???

printf(“%lu”, *metaMetaNum);
???
printf(“%lu”, **metaMetaNum);
???
printf(“%lu”, *num);
???
```

C: Warm-up - Pointers and Arrays

```
int num = 5; //num == 5
int *metaNum = &num; //metaNum == adr. of num
int **metaMetaNum = &metaNum; //???

printf(“%lu”, *metaMetaNum);
???
printf(“%lu”, **metaMetaNum);
???
printf(“%lu”, *num);
???
```

C: Warm-up - Pointers and Arrays

```
int num = 5; //num == 5
int *metaNum = &num; //metaNum == adr. of num
int **metaMetaNum = &metaNum;
//metaMetaNum == adr. of metaNum
printf(“%lu”, *metaMetaNum);
???
printf(“%lu”, **metaMetaNum);
???
printf(“%lu”, *num);
???
```

C: Warm-up - Pointers and Arrays

```
int num = 5; //num == 5
int *metaNum = &num; //metaNum == adr. of num
int **metaMetaNum = &metaNum;
//metaMetaNum == adr. of metaNum
printf(“%lu”, *metaMetaNum);
adr. of num
printf(“%lu”, **metaMetaNum);
???
printf(“%lu”, *num);
???
```

C: Warm-up - Pointers and Arrays

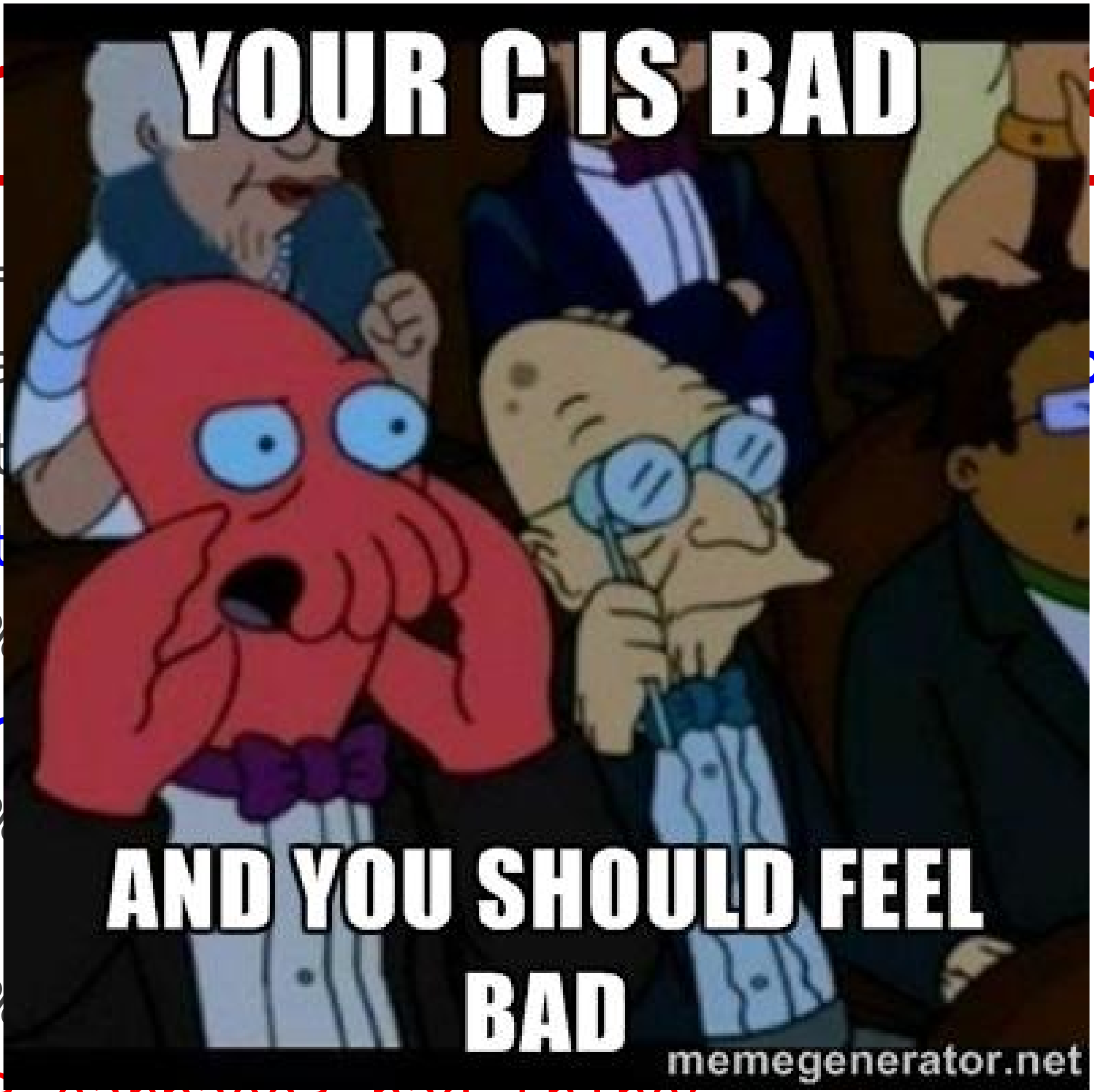
```
int num = 5; //num == 5
int *metaNum = &num; //metaNum == adr. of num
int **metaMetaNum = &metaNum;
//metaMetaNum == adr. of metaNum
printf(“%lu”, *metaMetaNum);
adr. of num
printf(“%lu”, **metaMetaNum);
5
printf(“%lu”, *num);
???
```

C: Warm-up - Pointers and Arrays

```
int num = 5; //num == 5
int *metaNum = &num; //metaNum == adr. of num
int **metaMetaNum = &metaNum;
//metaMetaNum == adr. of metaNum
printf(“%lu”, *metaMetaNum);
adr. of num
printf(“%lu”, **metaMetaNum);
5
printf(“%lu”, *num);
segfault? garbage? bad things.
```

C: War ays

```
int num =
int *meta                                   of num
int **met
//metaMet
printf("%
adr. of n
printf("%
5
printf("%
segfault? garbage? bad things.
```



C: Arrays

You can initialize arrays as follows:

```
int arr[] = {1, 2, 3, 4};
int arry[3];
arry[0] = 1;
arry[1] = 2;
```

When you increment the pointer, the pointer will increment by the ***size of the data type specified***.

```
*(arr1 + 1) == arr[1]; // True
```

C: Strings

Strings are `char` arrays that *always end in* `'\0'` (null).

```
char * hello = "hello world";
hello[1] == 'e';    // True
hello[11] == '\0'; // True
```

C: What does this do?

```
void bbq(char* wow)
{
    printf("{");
    for (; *(wow + 1) != '\0'; wow++) {
        printf("%c, ", *wow);
    }
    printf("%c}\n", *(wow)++);
}
char lol[] = "Mystery";
bbq(lol);
???
```

C: What does this do?

```
void bbq(char* wow)
{
    printf("{");
    for (; *(wow + 1) != '\0'; wow++) {
        printf("%c, ", *wow);
    }
    printf("%c}\n", *(wow)++);
}
char lol[] = "Mystery";
bbq(lol);
{M, y, s, t, e, r, y}
```

C: Pointers

```
char string[] = "randyisdandy";
char *ptr = string;

printf("*ptr: %c\n", *ptr);
???
printf("%c\n", *ptr++);
???
printf("%c\n", ++*ptr);
???
printf("%c\n", *++ptr);
???
```

C: Pointers

```
char string[] = "randyisdandy";
char *ptr = string;

printf("*ptr: %c\n", *ptr);
r
printf("%c\n", *ptr++);
???
printf("%c\n", ++*ptr);
???
printf("%c\n", *++ptr);
???
```

C: Pointers

```
char string[] = "randyisdandy";
char *ptr = string;

printf("*ptr: %c\n", *ptr);
r
printf("%c\n", *ptr++);
r
printf("%c\n", ++*ptr);
???
printf("%c\n", *++ptr);
???
```

C: Pointers

```
char string[] = "randyisdandy";
char *ptr = string;

printf("*ptr: %c\n", *ptr);
r
printf("%c\n", *ptr++);
r
printf("%c\n", ++*ptr);
b
printf("%c\n", *++ptr);
???
```

C: Pointers

```
char string[] = "randyisdandy";
char *ptr = string;

printf("*ptr: %c\n", *ptr);
r
printf("%c\n", *ptr++);
r
printf("%c\n", ++*ptr);
b
printf("%c\n", *++ptr);
n
```

C: Pointers

```
char string[] = "randyisdandy";
char *ptr = string; //sizeof(char) == 1

printf("*ptr: %c\n", *ptr); // r
printf("%c\n", *ptr++); // r
printf("%c\n", ++*ptr); // b
printf("%c\n", *++ptr); // n

int *ohno = string; //sizeof(int) == 4
printf("%c\n", *++ohno);
???
```

C: Pointers

```
char string[] = "randyisdandy";
char *ptr = string;

printf("*ptr: %c\n", *ptr); // r
printf("%c\n", *ptr++); // r
printf("%c\n", ++*ptr); // b
printf("%c\n", *++ptr); // n

int *ohno = string;
printf("%c\n", *++ohno);
y // because sizeof(int) == 4
```